EECS 280 Midterm Exam
Spring 2019
Questions Packet

This is a closed-book exam. You may use one note sheet, 8.5"x11", double-sided, with your name on it. This booklet contains multiple-choice questions, reference code for the written portion, and space for scratch work.

Read the entire exam through before you begin working. Work on those problems you find easiest first. Read each question carefully, and note all that is required of you. Assume all code is in standard C++11, and use only standard C++11 in your solutions.

Instructions:

- **Record ALL your solutions in the separate solutions packet.**
- This questions packet **WILL** be turned in. However, **NOTHING** written inside will be graded. You must record your answers in the separate solutions packet.
- Turn in both this booklet and the written-portion exam book. One will not be accepted without the other. **DO NOT turn in your note sheet**.
- There is no penalty for wrong answers. It is to your benefit to record an answer to each question, even if you're not sure what the correct answer is.
- Throughout the exam, assume all necessary `#include` headers and the `using namespace std;` directive are present unless otherwise directed.
- You do not need to verify `REQUIRES` clauses with `assert` unless instructed to do so.
- **The last several pages include space for scratch work** (including the last piece of paper, which you may tear out).

Problem 0: Short Answers (28 Points)

0a) True/False (10 points) 7:17 - 7:21 4分

T ✓ **1.** Declaring a reference variable creates another name for an existing object in memory.

T ✓ **2.** It is unsafe for a function to return a local variable by reference. return &a

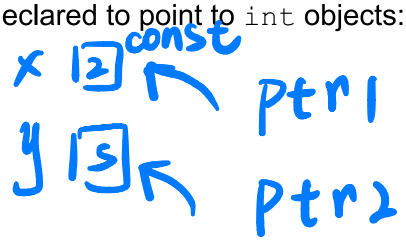
F ✓ **3.** The declaration `int * const ptr;` indicates that `ptr` may not be used to change the value stored in the `int` object it points to.

T ✓ **4.** For function parameters, the compiler changes any array parameter to a pointer parameter.

F ✓ **5.** If `str` is a pointer of type `char *` used to traverse the characters in a C-style string, the end of the string is reached once `str` becomes a null pointer. "\0"

F ✓ **6.** Assuming the following pointers are declared to point to `int` objects:

```
int x = 2;
int y = 5;
const int *ptr1 = &x;
int *ptr2 = &y;
```



Then the assignment `*ptr2 = *ptr1` is prohibited by the compiler. X

F ✓ **7.** Because struct member variables are public by default, directly accessing the member variables of a C-Style ADT (without calling an ADT function) is not considered breaking the interface.

F ✓ **8.** Private member variables for a class may only be accessed in the scope of that class or in the scope of any of its derived classes. X

T ✓ **9.** A base class constructor is always called during the creation of a derived class object.

T ✓ **10.** Assuming the `Chicken` class is derived from `Bird`, and `b` and `c` are declared as:

```
Bird b;
Chicken c;
```

Both the static and dynamic type of `b` are always `Bird`, even if a statement such as `b = c` is executed.

*b *c

0b) What is the output? (12 points)

Consider the following functions: 7:25 - 7:30

```
int * apple(int a, int *b) {
  a += 1;         2       3
  b += 1;
  return &a;
}

int * banana(int &c, int *d) {
  c += 1;          2       3
  *d += 1;
  return &c;
}
```

Consider each code snippet below. Write the **output** of each of the code snippets in the solutions packet. If the code does not compile, write **"compile error"**. If it compiles but results in undefined behavior at runtime, write **"undefined"**. If it compiles and runs but prints nothing, write **"no output"**. Do not worry about precise whitespace or newlines - we will ignore whitespace when grading.

1.
```
int x = 2; int y = 3;
apple(x, &y);
cout << x << " " << y << endl;
        2
```
2 - ~~4~~ 2 - 3

2.
```
int x = 2; int y = 3;
banana(x, &y);
cout << x << " " << y << endl;
```
3 - 4 ✓

3.
```
int x = 2; int y = 3;
int *z = apple(x, &y);
cout << *z << endl;
```
undefined. ✓

4.
```
int x = 2; int y = 3;
int *z = banana(x, &y);
cout << *z << endl;
```
3 ✓

5.
```
               3          2
int arr[2] = {2, 3};
if ( arr == banana(*arr, arr) ) { cout << "equal "; }
cout << arr[0] << " " << arr[1] << endl;
```
equal 4-3

equal 3✗-4

6.
```
int arr[2] = {2, 3};
if ( arr == banana(arr, *arr) ) { cout << "equal "; }
cout << arr[0] << " " << arr[1] << endl;
```
compile error ✓

0c) Does it compile? (6 points)

7:38 -7:41

Consider the following `RollerCoaster` class:

```
class RollerCoaster {
private:
  string name;
  int height_to_ride;
  int intensity_level;

public:
  RollerCoaster(const string &name);
  RollerCoaster(const string &name, int height, int intensity);

  bool can_ride_coaster(int my_height) const;

  int get_intensity_level() const;

  void increase_intensity_level();

  void print() const;
};
```

Determine whether the following code snippets contain a compile error. Consider each snippet individually. Assume each snippet is contained in a function outside the `RollerCoaster` class, e.g. `main`. Write your answers in the separate solutions packet by writing "**error**" or "**no error**".

1.
```
RollerCoaster r1("Millenium Force", 50);
bool can_ride = r1.can_ride_coaster(48);
```
error ✓

2.
```
RollerCoaster r2("Top Thrill Dragster", 52, 10);
const RollerCoaster * r_ptr = &r2;
r_ptr->increase_intensity_level();
```
error ✓

3.
```
const RollerCoaster r3("Valravn");
r3.print();
```
no error ✓

4.
```
RollerCoaster r4("Rougarou");
while (r4.intensity_level < 100) {
  r4.increase_intensity_level();
}
```
error ✓

no defaul constructor.

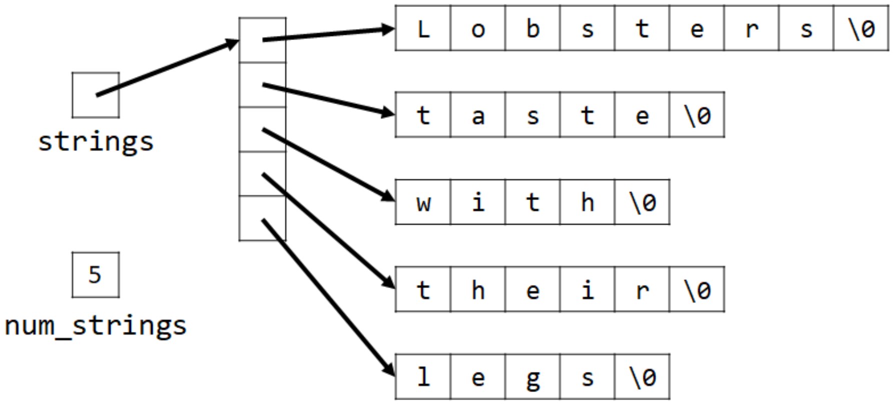
5.
```
RollerCoaster coasters[5];
coasters[0].print();
```
no error ✗ error

6.
```
RollerCoaster r6("The Outlaw");
RollerCoaster * const r_ptr = &r6;
r_ptr->increase_intensity_level();
```
no error ✓

Problem 1: Strings and I/O (23 Points)

1a) (10 points)

Consider a structure that represents a list of strings as a pointer to an array of c-style strings:



Implement the `longest_string()` function according to the RME.

- You **MAY NOT** use any classes or functions from the standard library (e.g. `std::string` or `strlen()`). This restriction only applies for part 1a.

```
// REQUIRES: num_strings > 0
//           strings points to an array of c-style strings
//           All C-style strings are valid strings
// EFFECTS:  Returns a pointer to the longest C-style string in the
//           given strings structure. You may break ties either way.
//
// EXAMPLE:  Given the sample structure shown, the function returns
//           a pointer to the C-style string containing "Lobsters".
const char* longest_string(const char *strings[], int num_strings) {
  // Implement this function in the solutions packet
}
```

1b) (5 points) Implement the `limited_copy()` function according to the RME.

```
// REQUIRES: limit >= 0
// EFFECTS:  Reads words (as strings) from the input stream is and
//           writes them to the output stream os until either limit
//           words have been written, or there is no more input. Each
//           word is followed by a single space in the output.
// NOTE:     You should use std::string for this problem.
void limited_copy(istream &is, ostream &os, int limit) {
  // Implement this function in the solutions packet
}
```

1c) (8 points) Consider the program below, which takes an output filename, an integer N number of words to be copied, and a list of candidate input filenames (in that order). It selects the input file with the longest filename (including the file extension) and writes the first N words from the input file to the output file. Each output word should be followed by a space. If N is greater than the number of words in the selected input file then all the words should be copied.

Example: If the program is compiled to main.exe and run as:

```
$./main.exe output.txt 4 hello.cpp Matrix.h lobster.txt
```

The file lobster.txt would be selected to read from. Assume it contains:

lobster.txt	Did you know lobsters can live as long as 100 years?

Then the first 4 words would be written to output.txt:

output.txt	Did you know lobsters

Write an implementation for the main function for this program in the solutions packet.

- Your implementation **MUST** use the longest_string and limited_copy functions.
- You may break ties for the longest filename either way (i.e. just use longest_string).
- **Error checking:** Your implementation **MUST** check whether there are enough arguments - the output filename, the number of words to copy, and at least one candidate input filename. If there are not enough, return 1 from main. You **DO NOT** need to check for any other errors, such as whether files open successfully.

Portions of the RMEs and the signatures for longest_string and limited_copy are repeated here for convenience:

```
// REQUIRES: num_strings > 0
//           strings points to an array of c-style strings
//           All C-style strings are valid strings
// EFFECTS:  Returns a pointer to the longest C-style string in the
//           given strings structure. You may break ties either way.
const char* longest_string(const char *strings[], int num_strings);

// EFFECTS:  Reads words (as strings) from the input stream is and
//           writes them to the output stream os until either limit
//           words have been written, or there is no more input. Each
//           word is followed by a single space in the output.
void limited_copy(istream &is, ostream &os, int limit);
```

Problem 2: Arrays and Pointers (10 Points)

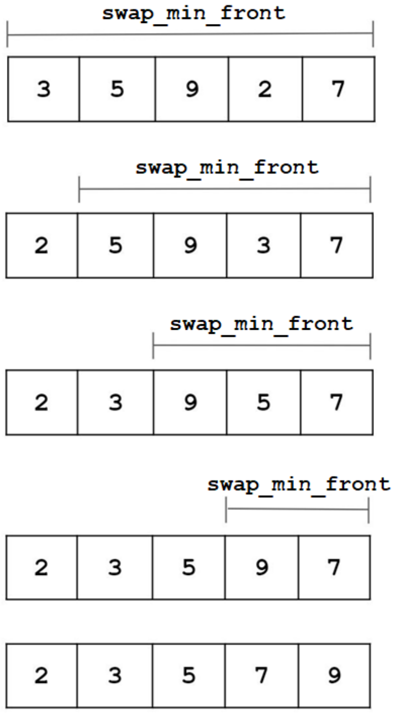
2a) (4 points) Implement the swap_min_front() function according to the RME, and using the two helper functions provided below. Note the two helper functions below the RME.

```
// REQUIRES: arr points to an array with size n
//           n >= 0
// EFFECTS:  Moves the minimum value to the leftmost position by
//           swapping it with the current leftmost element. If
//           there is a tie for the min, moves the leftmost one.
void swap_min_front(int *arr, int n){
  // Implement this function in the solutions packet
}
```

In your implementation, you **MUST** use two helper functions: swap and find_min.

```
// EFFECTS: Swaps the values of objects pointed to by a and b.
void swap(int *a, int *b);

// REQUIRES: arr points to an array (or subarray) with size n
// EFFECTS:  Returns a pointer to the minimum value in arr.
//           If there is a tie for the min, finds the leftmost one.
int * find_min(int *arr, int n);
```



2b) (6 points) Repeated applications of the swap_min_front function can be used to sort an array. The trick is to start by calling it on the whole array, then call it on successively smaller subarrays, as shown in the picture at right. **Take some time to understand this algorithm before writing code.**

Implement the ascending_sort() function **using this algorithm**. Your implementation **MUST** use swap_min_front. You are **NOT** allowed to call any other functions.

```
// REQUIRES: arr points to an array
//           with size n
// EFFECTS:  sorts arr
void ascending_sort(int *arr, int n) {
  // Implement in the solutions packet
}
```

Problem 3: Structs and C-Style ADTs (16 points)

Use the following C-Style ADTs for this problem. We recommend you briefly skim them before answering questions. You may assume all necessary #include headers and the using namespace std; directive are present.

The Artist and Festival C-style ADTs (implemented with structs) are used to represent the wildly popular local music festival "EECSapalooza".

```
struct Artist {
  string name;
  string venue;
  bool sold_out;
};

// REQUIRES: a_ptr points to a Artist
// MODIFIES: *a_ptr
// EFFECTS: Initializes the given Artist.
void Artist_init(Artist *a_ptr, const string &name,
                 const string &venue, bool sold_out);

// REQUIRES: a_ptr points to a valid Artist
// EFFECTS: Returns the name of the given Artist.
string Artist_name(const Artist *a_ptr);

// REQUIRES: a_ptr points to a valid Artist
// EFFECTS: Returns the venue at which the Artist is performing.
string Artist_venue(const Artist *a_ptr);

// REQUIRES: a_ptr points to a valid Artist
// EFFECTS: Returns whether the given Artist is sold out.
bool Artist_is_sold_out(const Artist *a_ptr);

// REQUIRES: a_ptr points to a valid Artist
// EFFECTS:  Prints a description of the given Artist to os.
// FORMAT:   If the Artist struct contains {"The Algorhythms", "BBB", false}:
//             "The Algorhythms performing now at BBB"
//           Or, if the Artist struct contains {"Lady Giga", "DOW", true}:
//             "Lady Giga performing now at DOW (SOLD OUT)"
// NOTE:     There is no newline printed after the output.
void Artist_print(const Artist *a_ptr, ostream &os){
      // Implement this function in part 3a
}
```

```
const int MAX_FESTIVAL_SIZE = 100;

struct Festival {
  Artist artists[MAX_FESTIVAL_SIZE]; // array to hold Artists
  int num_artists; // number of valid Artists currently in the array

  // INVARIANT: The first num_artists elements of artists contain
  //            valid Artist objects.
};

// REQUIRES: f_ptr points to a valid Festival
//           The number of artists is less than the max festival size
// EFFECTS:  Returns true if ALL of the artists performing at the festival
//           are sold out (this also includes the case where there are no
//           artists at the festival). Returns false otherwise.
bool Festival_is_sold_out(const Festival *f_ptr) {
  // Implement this function in part 3b
}


// REQUIRES: f_ptr points to a valid Festival
// MODIFIES: *f_ptr
// EFFECTS:  Adds an artist to the festival with the given name and venue.
//           Initially, an artist is not sold out.
void Festival_add_artist(Festival *f_ptr, const string &name_in,
                         const string &venue_in) {
  // Implement this function in part 3c
}
```

3a) (4 points) Implement the `Artist_print()` function in the solutions packet.

3b) (6 points) Implement the `Festival_is_sold_out()` function in the solutions packet.

3c) (6 points) Implement the `Festival_add_artist()` function in the solutions packet.

Problem 4: Inheritance and Polymorphism (22 Points)

Use the following classes for this problem. We recommend you briefly skim them before answering questions. You may assume all necessary `#include` headers and the `using namespace std;` directive are present.

```
class Toy {
private:
  int yearMade;

public:
  Toy(int year_in) : yearMade(year_in) { cout << "Toy Ctor" << endl; }

  int get_year_made() const { return yearMade; }

  virtual void play() = 0;
};

class Robot : public Toy {
private:
  string name;
  double battery;

public:
  // EFFECTS: Initializes a Robot with the given name and year, and an
  //          initial battery level of 100. Also prints "Robot ctor".
  Robot(const string &name_in, int year_in); // Implement in part 4a

  double get_battery_level() const { return battery; }

  void introduce() const { cout << "Hi, my name is " << name << endl; }

  virtual void sound_effect() const { cout << "beep boop" << endl; }

  // EFFECTS: Prints out the sound effect for this robot. If the robot
  //          was made before the year 1970, the battery level
  //          decreases by 20. Otherwise, it decreases by 10.
  void play() override; // Implement in part 4b
};

class FighterBot : public Robot {
public:
  // EFFECTS: Initializes a FighterBot with the given name and year.
  //          Also prints "FighterBot ctor".
  FighterBot(const std::string &name_in, int year_in);
  // ^^^ Implementation not shown, but you may assume it works correctly.

  void introduce() const { cout << "Hello. I will destroy you." << endl; }

  void sound_effect() const override { cout << "Pow! Pow!" << endl; }
};
```

4a) (3 points) Implement the `Robot()` constructor in the solutions packet according its RME.

- Your implementation **MUST** use a member initializer list where appropriate.

4b) (5 points) Implement the `play()` function in the solutions packet according its RME.

4c) (6 points) Determine whether the following code snippets contain a compile error. Consider each snippet individually. Write your answers in the separate solutions packet.

1.
   ```
   Robot rBot("ROB", 1985);
   Toy *tPtr = &rBot;
   tPtr->introduce();
   ```

2.
   ```
   Toy yoyo(1928);
   cout << yoyo.get_year_made() << endl;
   ```

3.
   ```
   Robot rBot("Baymax", 2014);
   FighterBot *fPtr = &rBot;
   cout << fPtr->get_battery_level() << endl;
   ```

4.
   ```
   FighterBot fBot("Terminator", 1984);
   Toy *tPtr = &fBot;
   cout << tPtr->get_year_made() << endl;
   ```

4d) (8 points) Consider the code below, which *compiles successfully and runs without errors or undefined behavior*. Assume functions work according to their RME. Write the **output** of each portion of the code in the solutions packet (or "**no output**" if no output is produced).

```
int main() {
```

1	`FighterBot megaBot("Megaman", 1987);`

2	`Robot *ptr = &megaBot;`
	`ptr->sound_effect();`

3	`ptr->introduce();`

4	`ptr->play();`
	`cout << ptr->get_battery_level() << endl;`

```
}
```

Usability Score: You've reached the end of the questions! Please remember to fill in the usability score on the front page of the solutions packet.

Space for Scratch Work

You may use this space for scratch work, but we will not grade anything here.

Space for Scratch Work

You may use this space for scratch work, but we will not grade anything here.

Space for Scratch Work

You may use this space for scratch work, but we will not grade anything here.